Application-Managed Flash

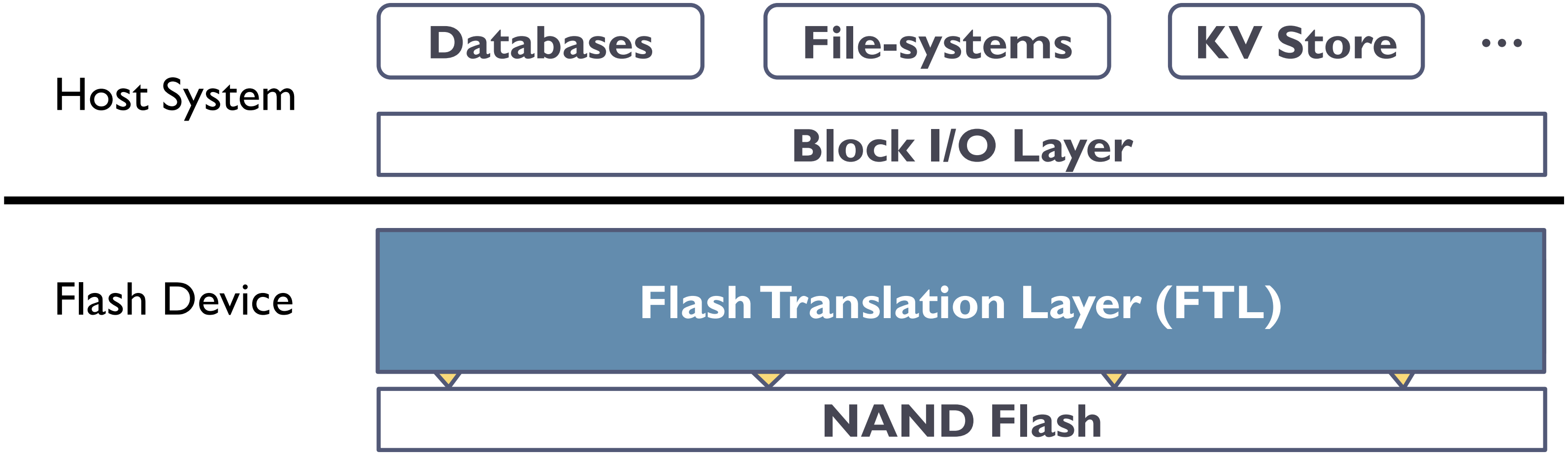
Sungjin Lee[*], **Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim**[†] **and Arvind**

[*]Inha University
Massachusetts Institute of Technology
[†]Seoul National University

Operating System Support for Next Generation Large Scale NVRAM (NVRAMOS)
October 20-21, 2016
(Presented at USENIX FAST '16)

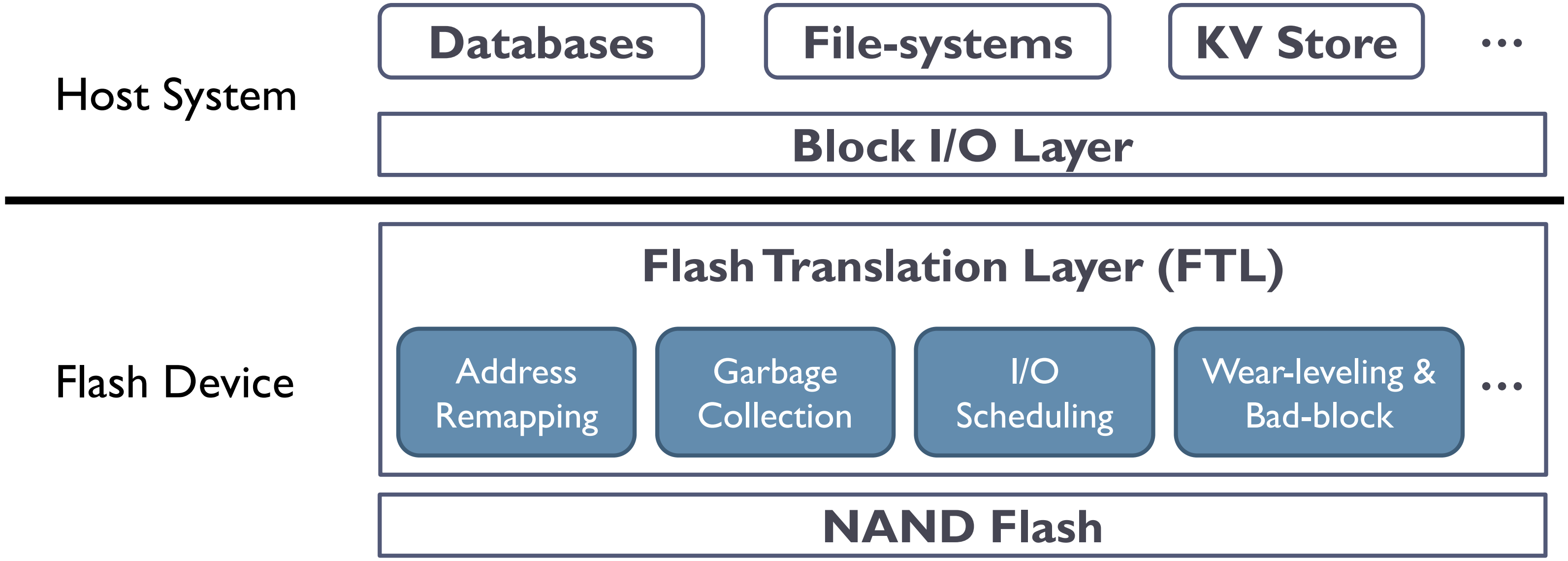
NAND Flash and FTL

- NAND flash SSDs have become the preferred storage devices in consumer electronics and datacenters
- FTL plays an important role in flash management
  - The principal virtue of FTL is providing interoperability with the existing block I/O abstraction



FTL is a Complex Piece of Software

- FTL runs complicated firmware algorithms to avoid in-place updates and manages unreliable NAND substrates


Host System

Databases

File-systems

KV Store

…

Block I/O Layer

Flash Device

Flash Translation Layer (FTL)

Address Remapping

Garbage Collection

I/O Scheduling

Wear-leveling & Bad-block

…

NAND Flash

- But, FTL is ***a root of evil*** in terms of HW resources and performance
  - Requires significant hardware resources (e.g., 4 CPUs / 1-4 GB DRAM)
  - Incurs extra I/Os for flash management (e.g., GC)
  - Badly affects the behaviors of host applications

Existing Approach

- **Improving FTL itself**
  - Better logical-to-physical address mapping and garbage collection algorithms

  → Limited optimization due to the lack of information
- **Optimizing FTL with custom interface**
  - Delivering system-level information to FTL for better optimization (e.g., file access pattern, hint when to trigger GC, and stream ID, …)

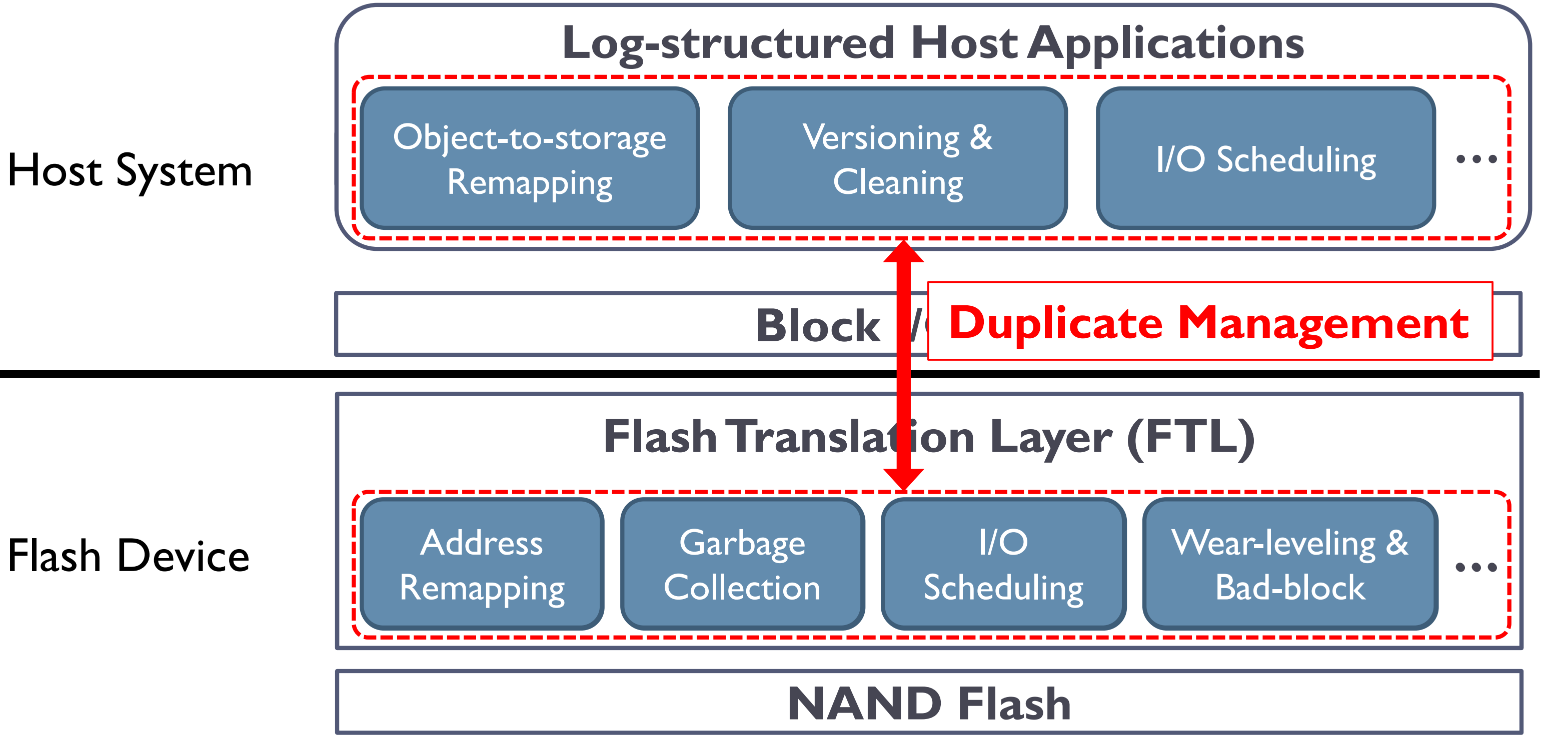
  → Special interfaces, hard for standardization, more functions added to FTL
- **Offloading host functions into FTL**
  - Moving some part of a file system to FTL (e.g., nameless writes and object-based flash storage)

  → More hardware resources and greater storage design complexity

Many efforts have been made to put more functions to flash storage devices

However,
Functionality of FTL is Mostly Useless

- Many host applications manage underlying storage in ***a log-like manner***, mostly avoiding in-place updates



Log-structured Host Applications

Host System

Object-to-storage Remapping

Versioning & Cleaning

I/O Scheduling

…

Block

Duplicate Management

Flash Translation Layer (FTL)

Flash Device

Address Remapping

Garbage Collection

I/O Scheduling

Wear-leveling & Bad-block

…

NAND Flash

- This duplicate management not only (1) incurs serious performance penalties but also (2) wastes hardware resources

Which Applications???

File Systems

SpriteLFS
WAFL
BlueSky
F2FS
NILFS
Btrfs
HDFS

Key-value Stores

LevelDB
RocksDB
LSM-Tree

Which
Applications
???

Databases

RethinkDB
Cassandra
BigTable
LogBase
MongoDB
Hyder

FlexVol

Storage Virtualization

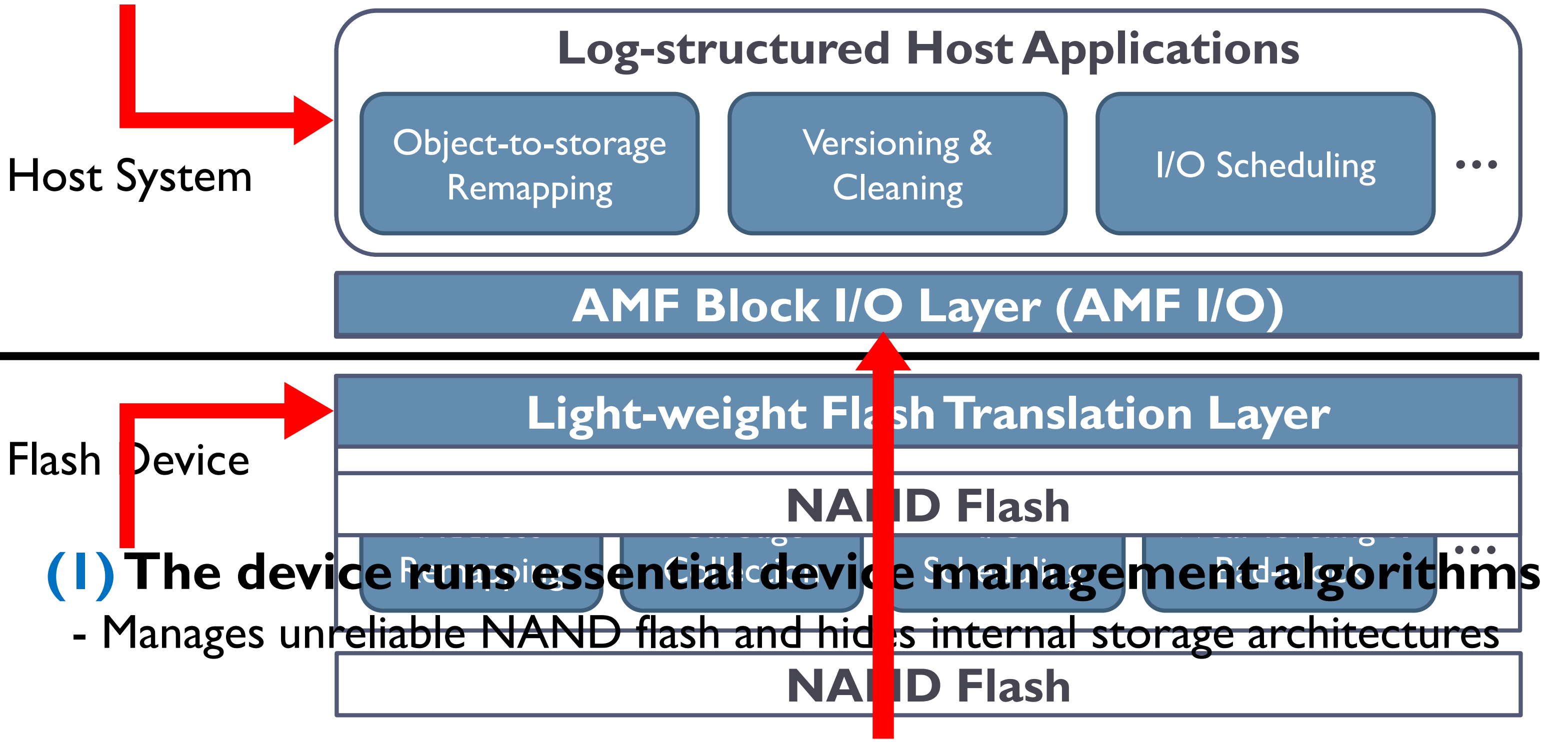
Question:

What if we removed FTL from storage devices and allowed host applications to directly manage NAND flash?

Application-Managed Flash (AMF)

(2) The host runs almost all of the complicated algorithms

- Reuse existing algorithms to manage storage devices



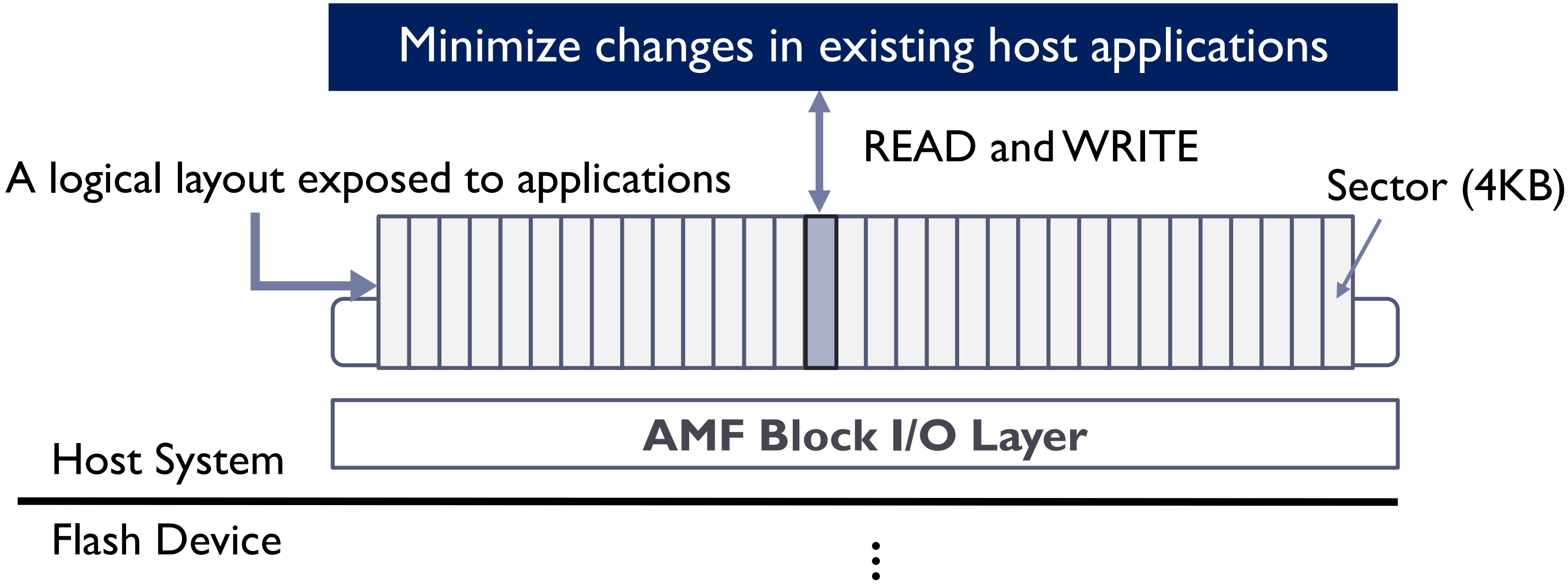
(1) The device runs essential device management algorithms

- Manages unreliable NAND flash and hides internal storage architectures

(3) A new AMF block I/O abstraction enables us to separate the roles of the host and the device

AMF Block I/O Abstraction (AMF I/O)

- AMF I/O is similar to a conventional block I/O interface
  - A linear array of fixed-size sectors (e.g., 4 KB) with existing I/O primitives (e.g., READ and WRITE)



Minimize changes in existing host applications

READ and WRITE

A logical layout exposed to applications

Sector (4KB)

AMF Block I/O Layer

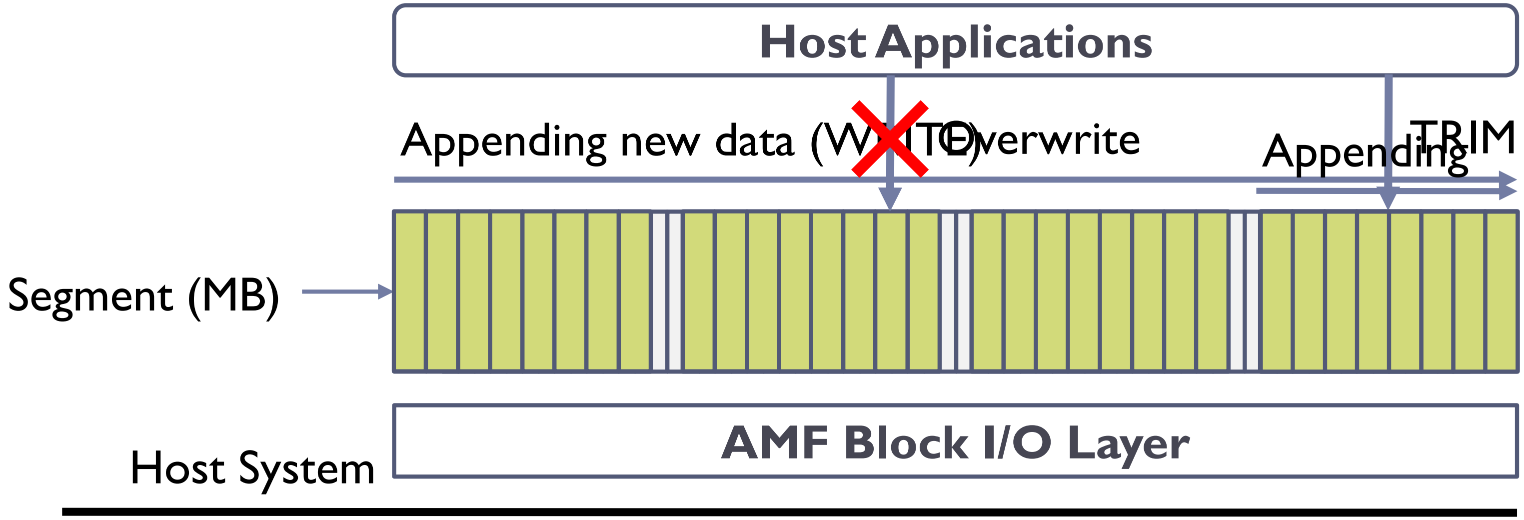
Host System

Flash Device

⋮

Append-only Segment

- Segment: a group of 4 KB sectors (e.g., several MB)
  - A unit of free-space *allocation* and free-space *reclamation*
- Append-only: overwrite of data is prohibited



Only sequential writes with no in-place updates
→ Minimize the functionality of the FTL

Case Study with AMF

File Systems

SpriteLFS
WAFL
BlueSky
F2FS
NILFS
Btrfs
HDFS

Key-value Stores

LevelDB
RocksDB
LSM-Tree

Which Applications ???

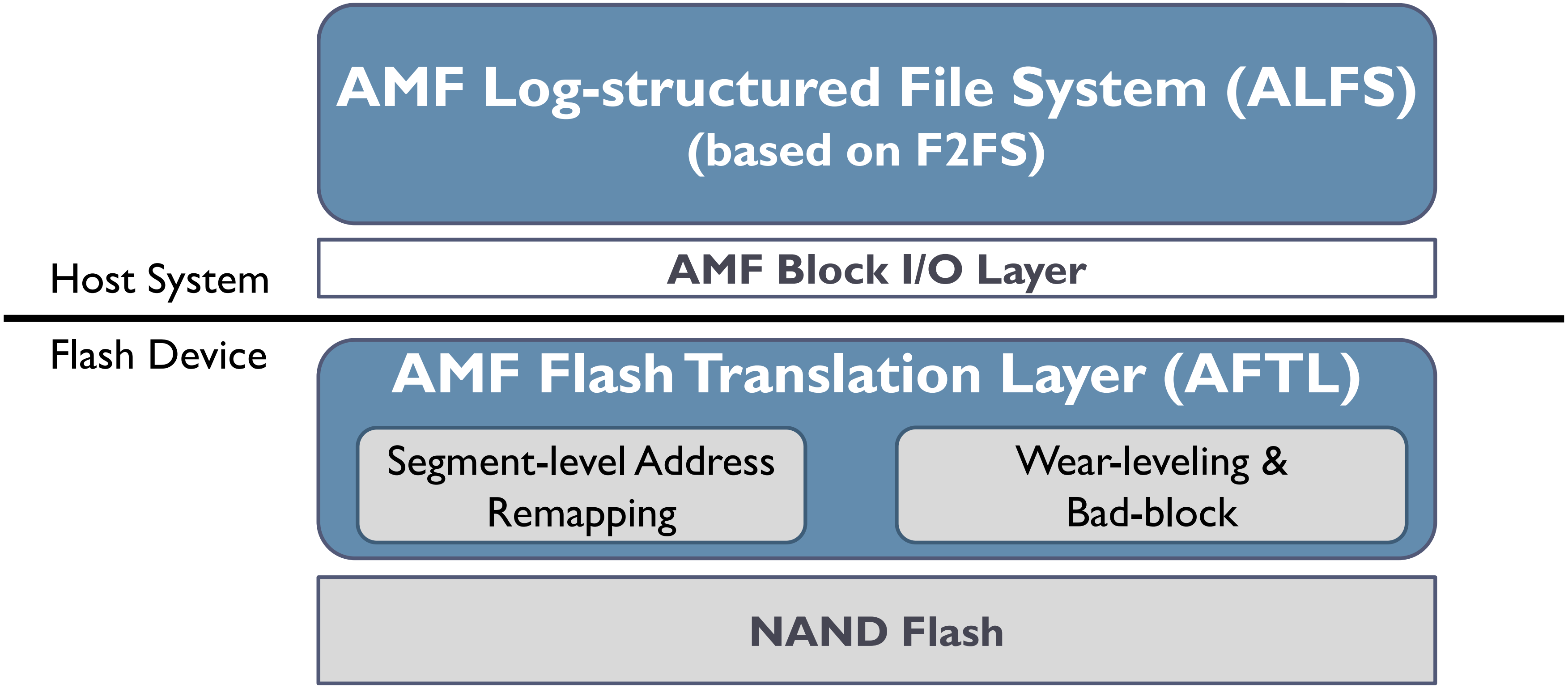
Databases

RethinkDB
Cassandra
BigTable
LogBase
MongoDB
Hyder

FlexVol

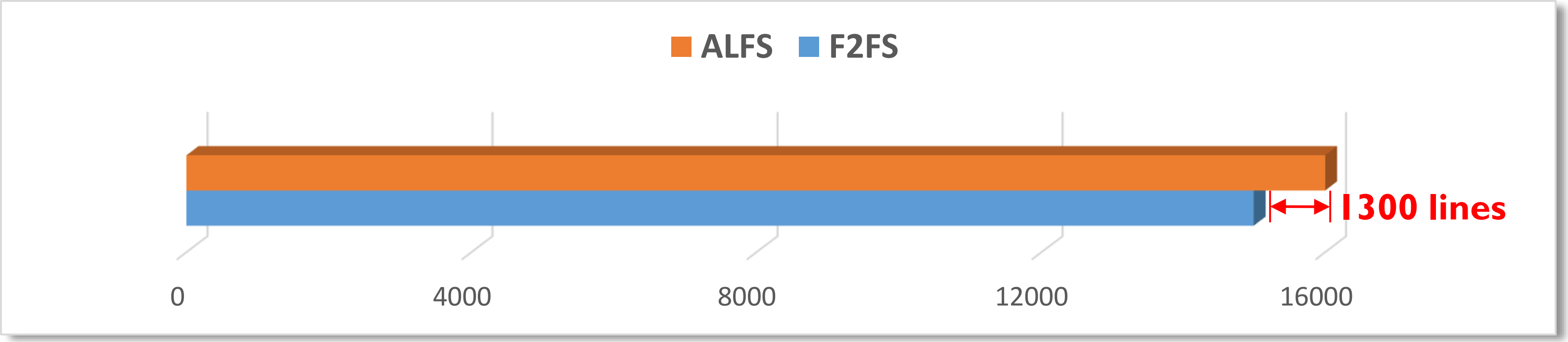
Storage Virtualization

Case Study with File System



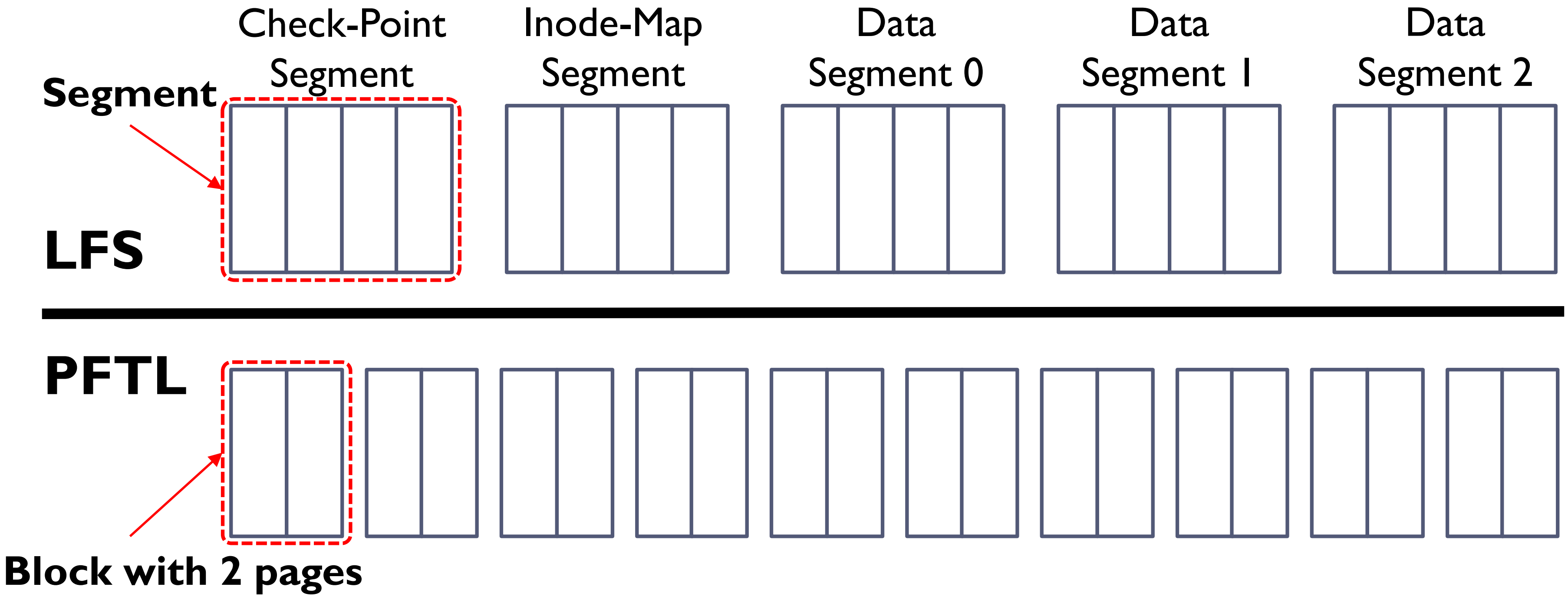
AMF Log-structured File System (ALFS)

- ALFS is based on the F2FS file system
- How did we modify F2FS for ALFS?
  - Eliminate in-place updates
    - F2FS overwrites check-points and inode-map blocks
  - Change the TRIM policy
    - TRIM is issued to individual sectors
- How many new codes were added?



<A comparison of source-code lines of F2FS and ALFS>

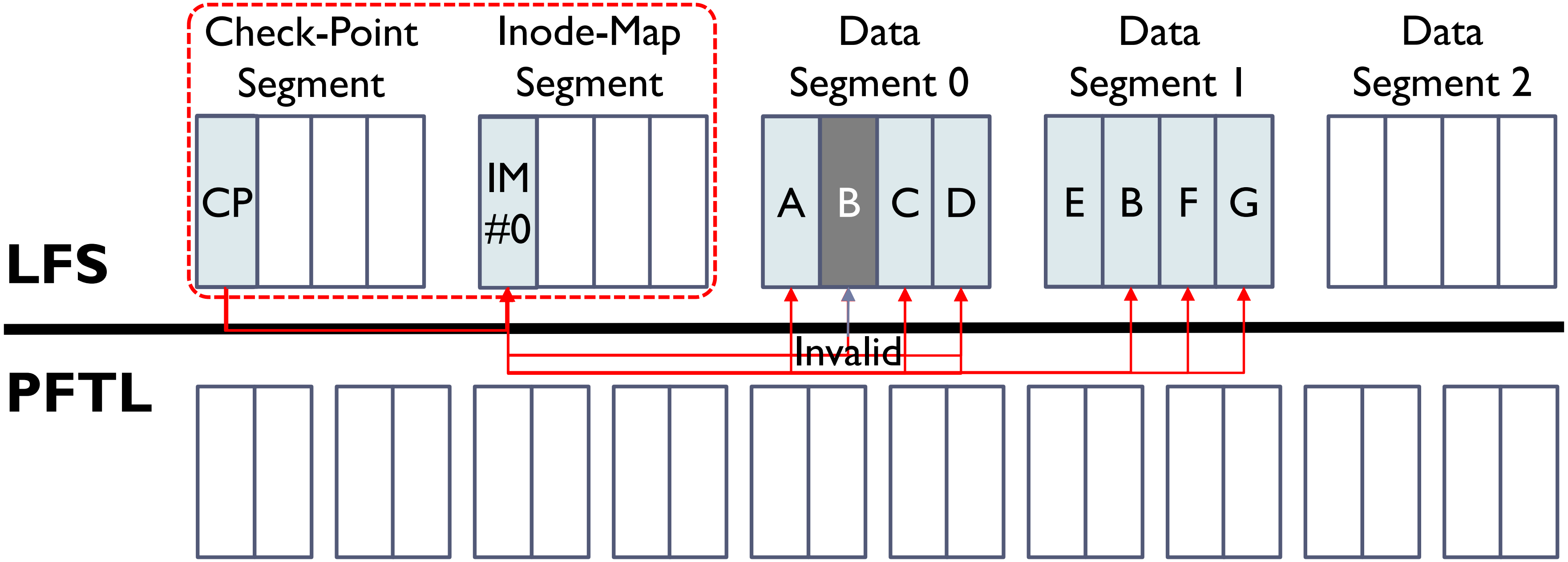
How Conventional LFS (F2FS) Works



*** PFTL: page-level FTL**

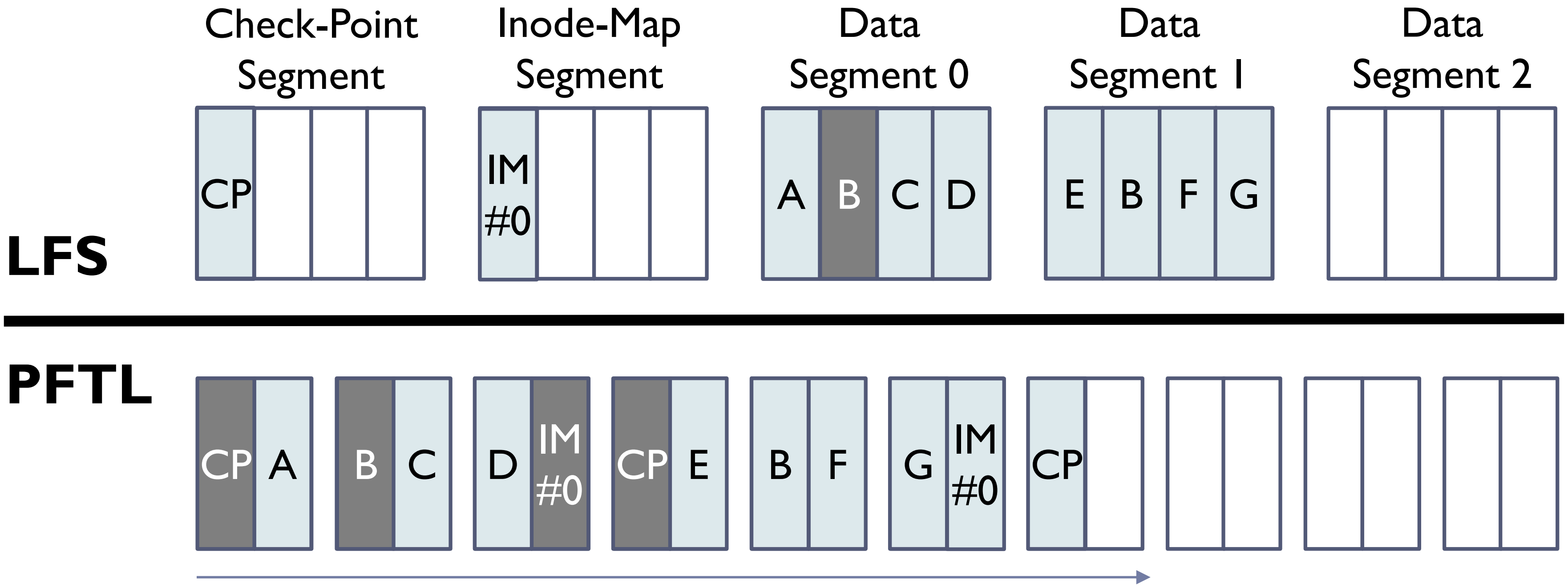
How Conventional LFS (F2FS) Works

Check-point and inode-map blocks are overwritten



* **PFTL: page-level FTL**

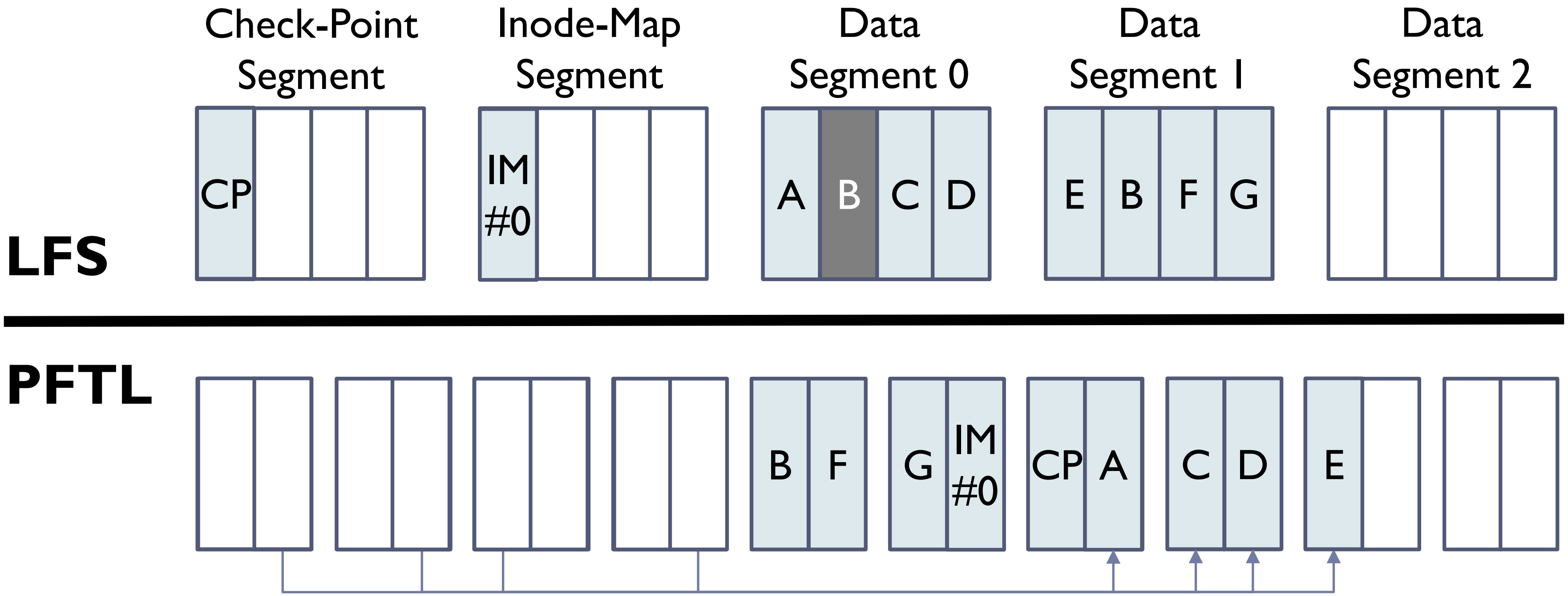
How Conventional LFS (F2FS) Works



The FTL appends incoming data to NAND flash

*** PFTL: page-level FTL**

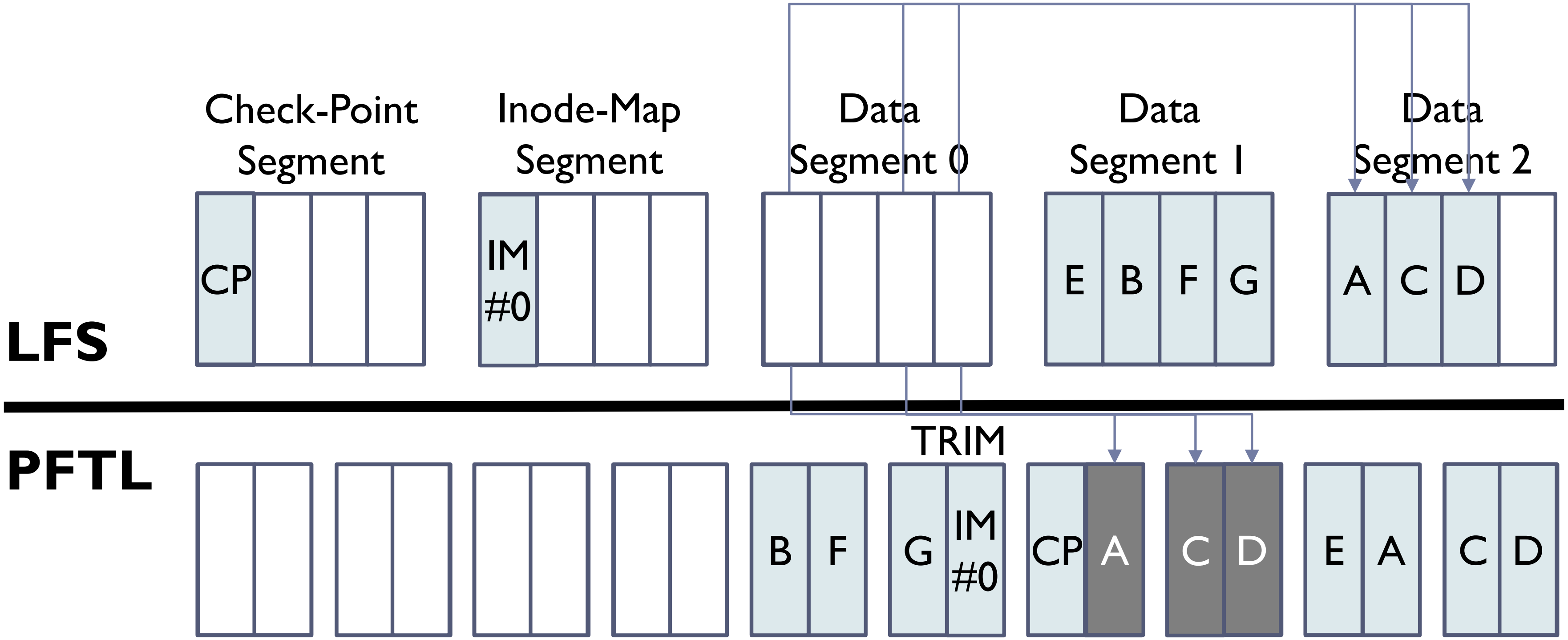
How Conventional LFS (F2FS) Works



The FTL triggers garbage collection: **4** page copies and **4** block erasures

*** PFTL: page-level FTL**

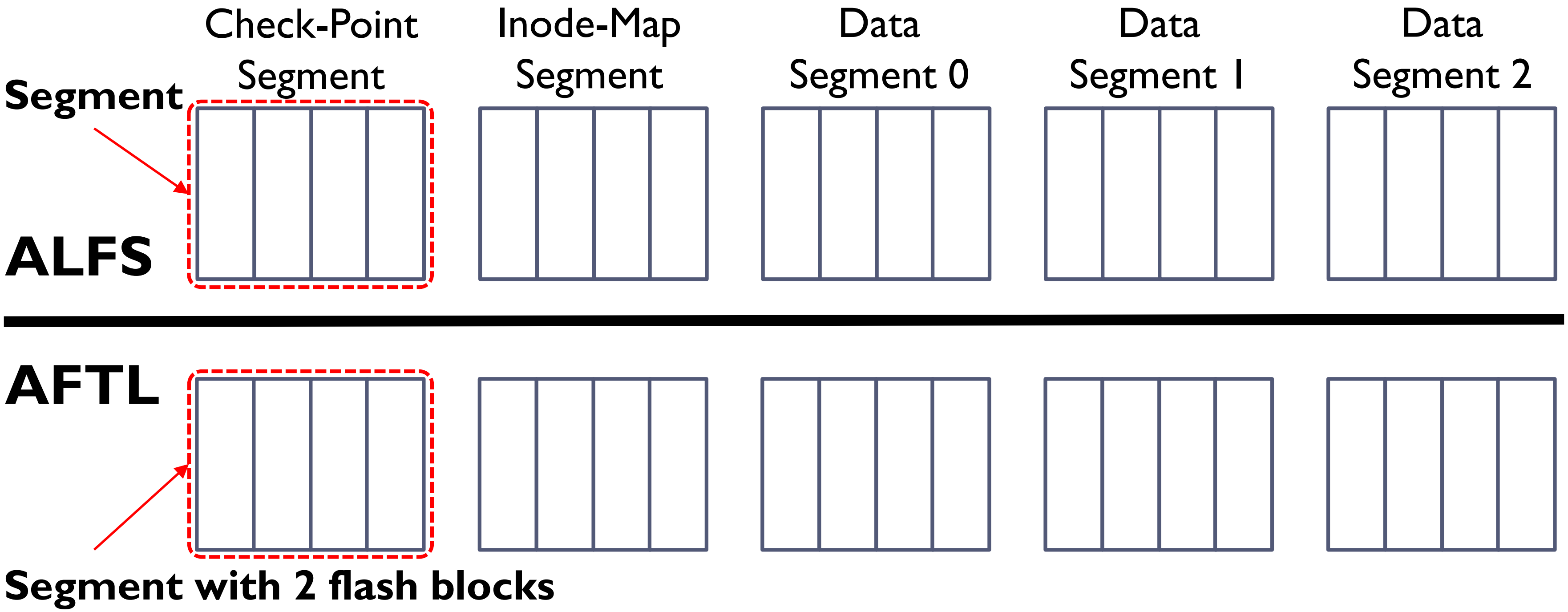
How Conventional LFS (F2FS) Works



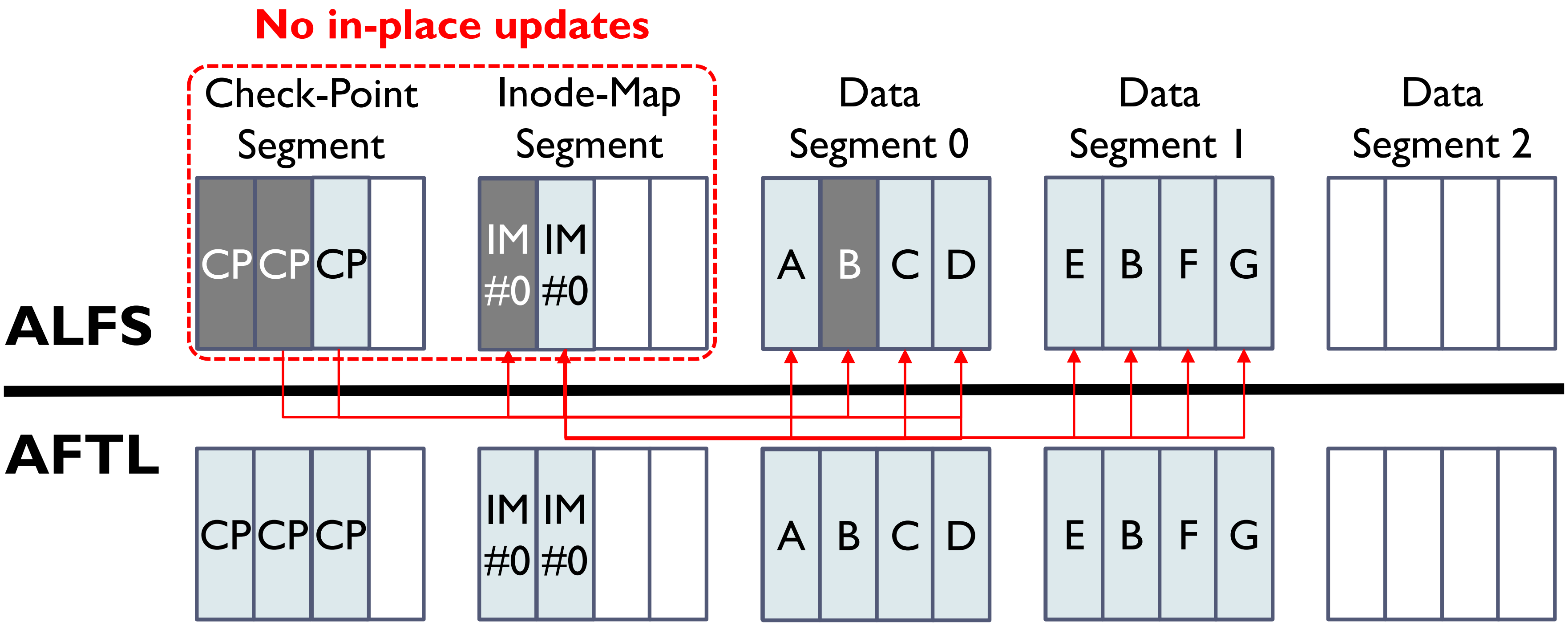
The LFS triggers garbage collection: **3** page copies

* **PFTL: page-level FTL**

How ALFS Works

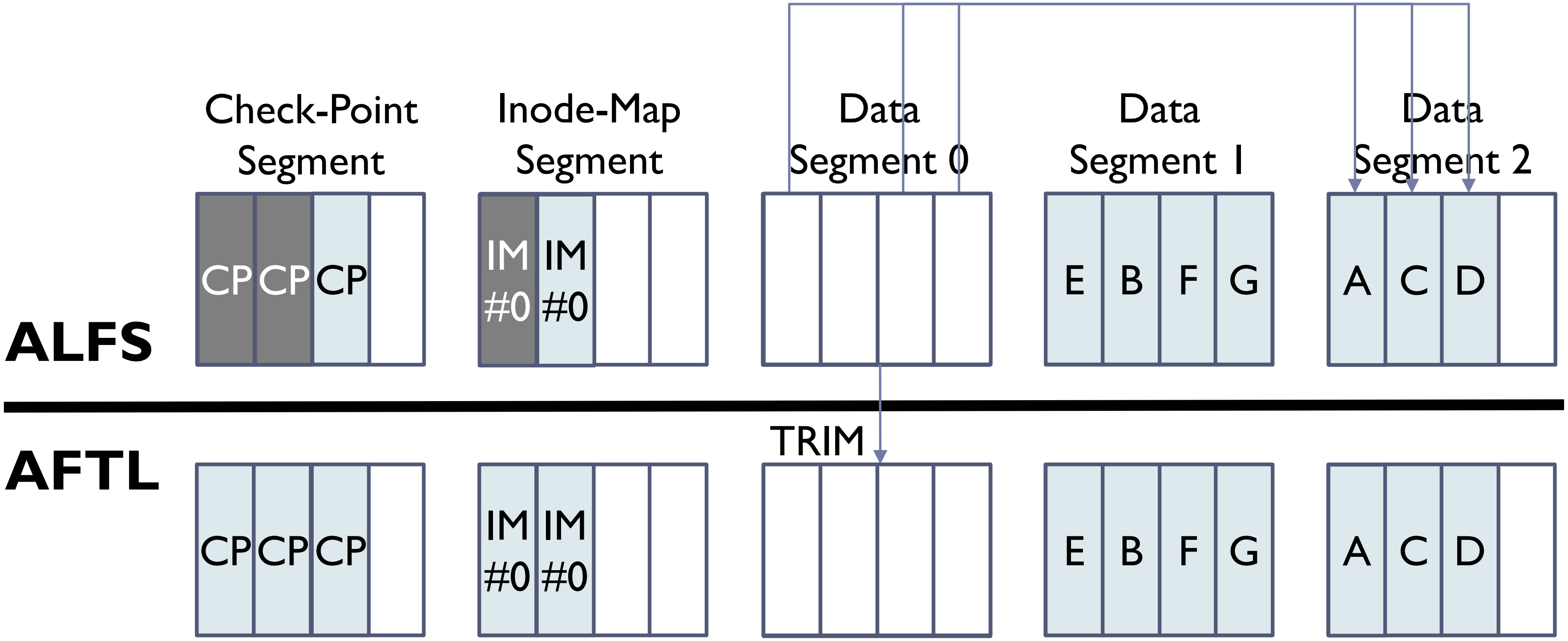


How ALFS Works



No obsolete pages – GC is not necessary

How ALFS Works



The ALFS triggers garbage collection: **3** page copies and **2** block erasures

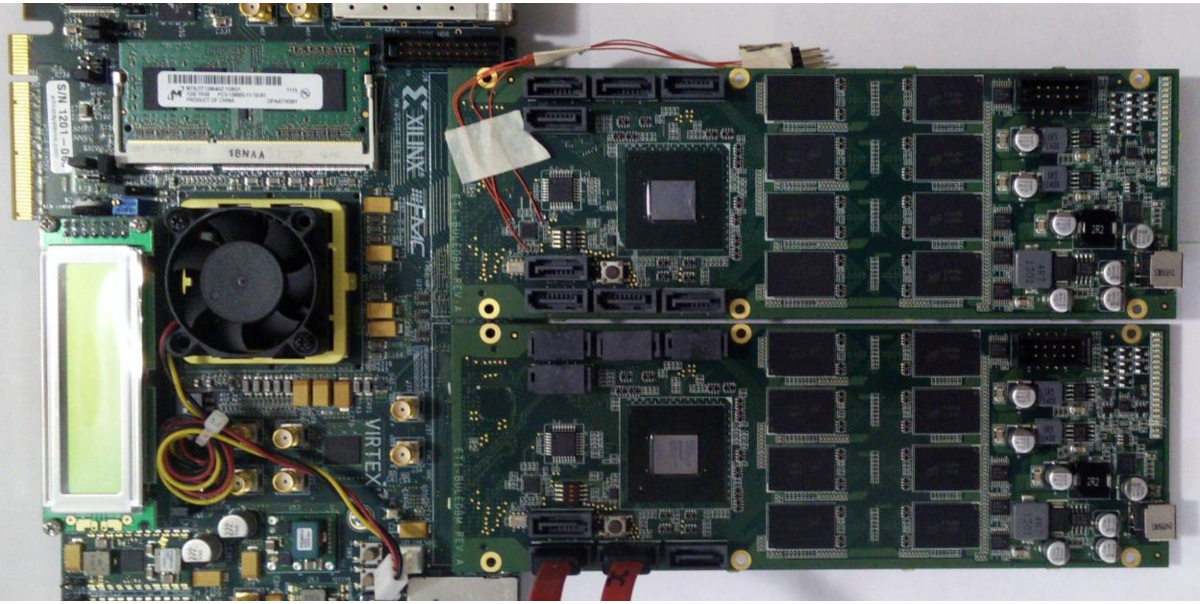
Comparison of F2FS and AMF

Duplicate Management

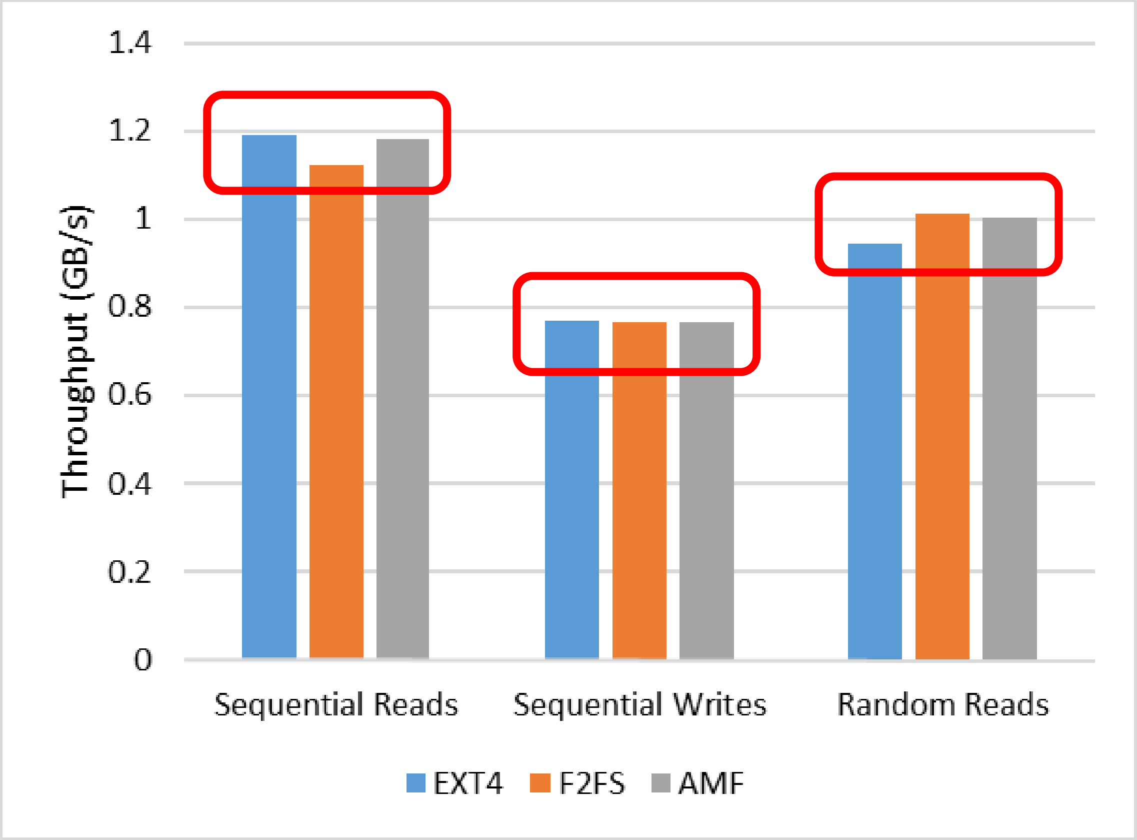
F2FS		AMF
File System	PFTL	File System
3 page copies	4 copies + 4 erasures	3 copies + 2 erasures
7 copies + 4 erasures		**3 copies + 2 erasures**

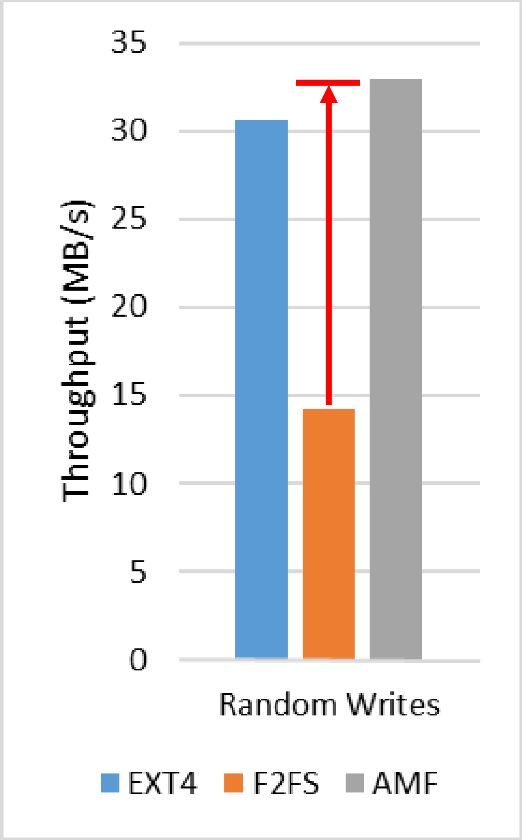
Experimental Setup

- Implemented ALFS and AFTL in the Linux kernel 3.13
- Compared AMF with different file-systems
  - Two file-systems: **EXT4** and **F2FS** with page-level FTL (PFTL)
- Ran all of them in our in-house SSD platform
  - BlueDBM developed by MIT



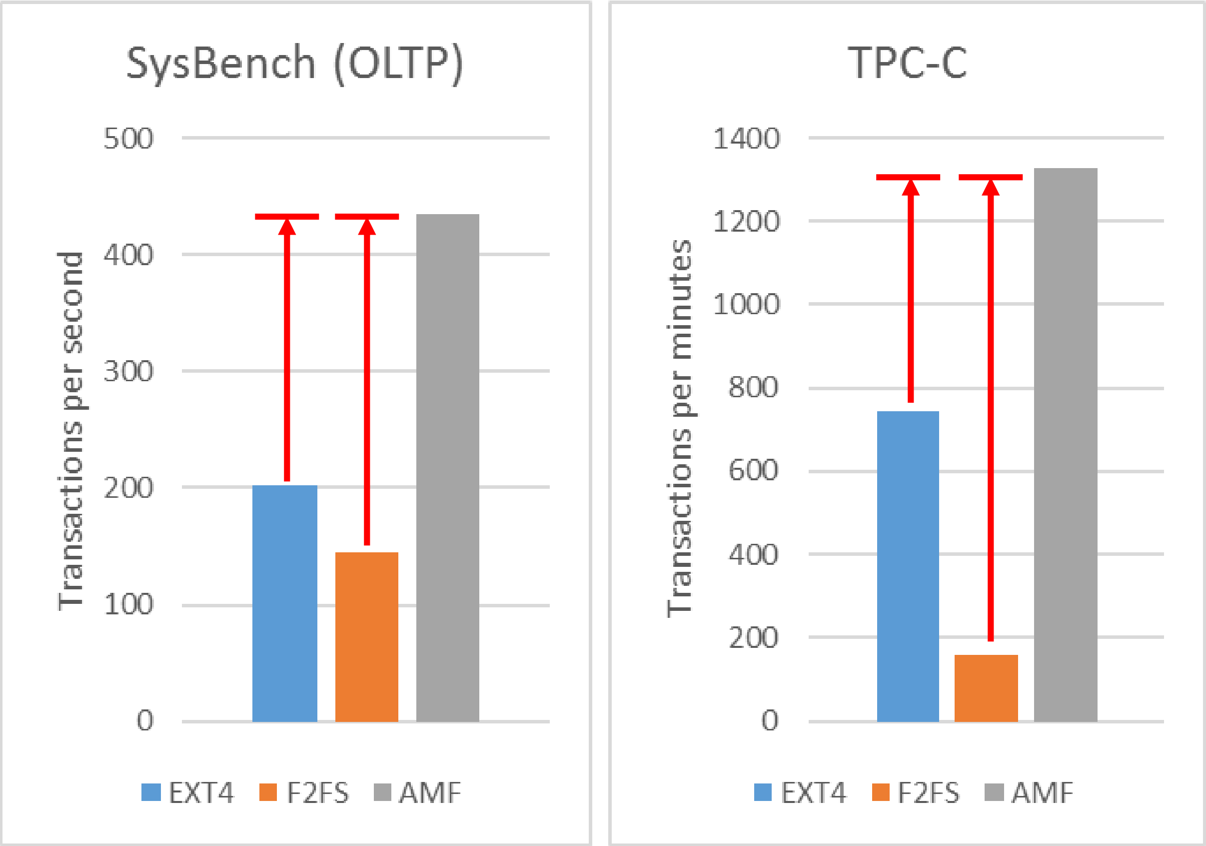
Performance with FIO





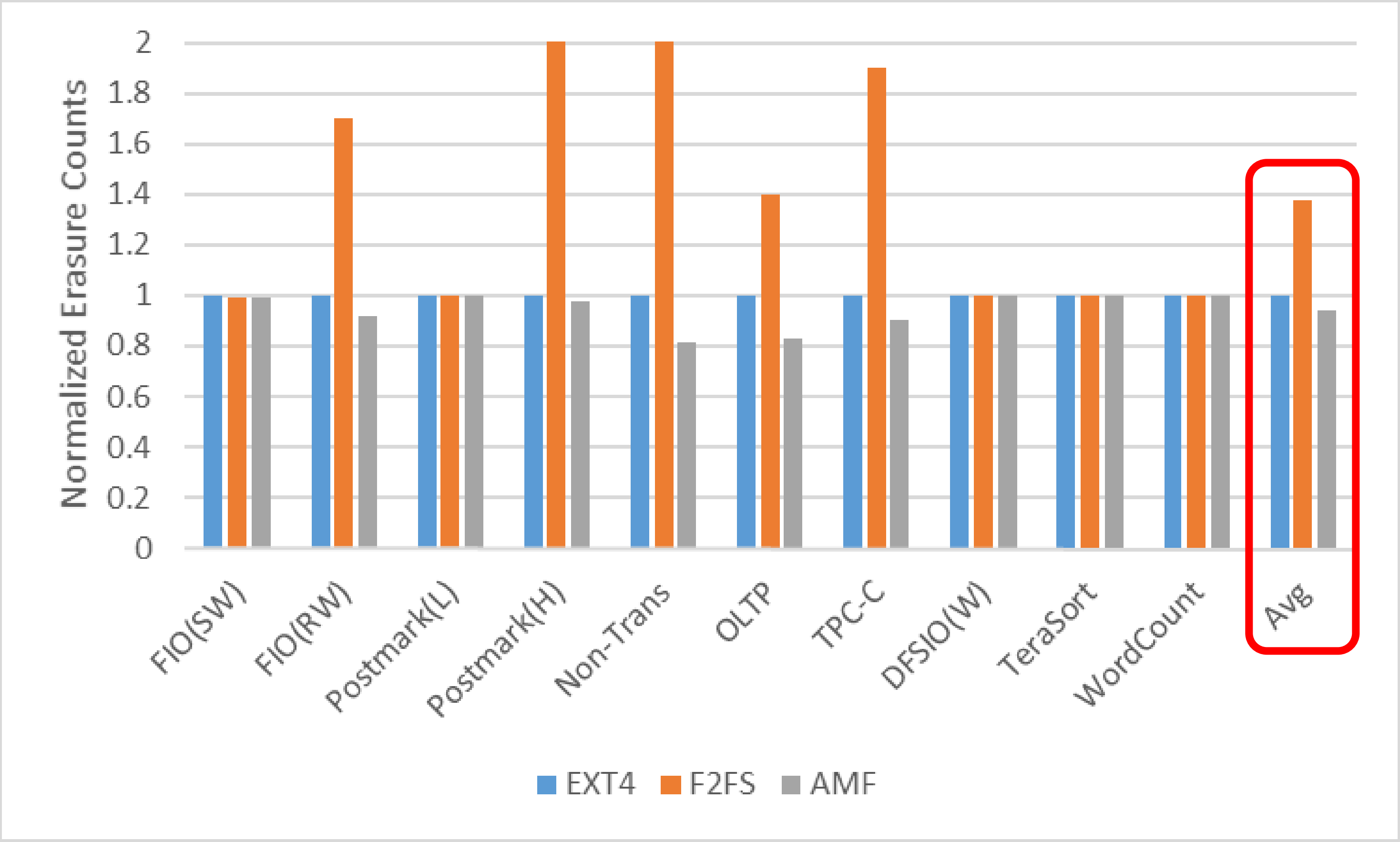
- For random writes, AMF shows better throughput
- F2FS is badly affected by the duplicate management problem

Performance with Databases



- AMF outperforms EXT4 with more advanced GC policies
- F2FS shows the worst performance

Erasure Counts



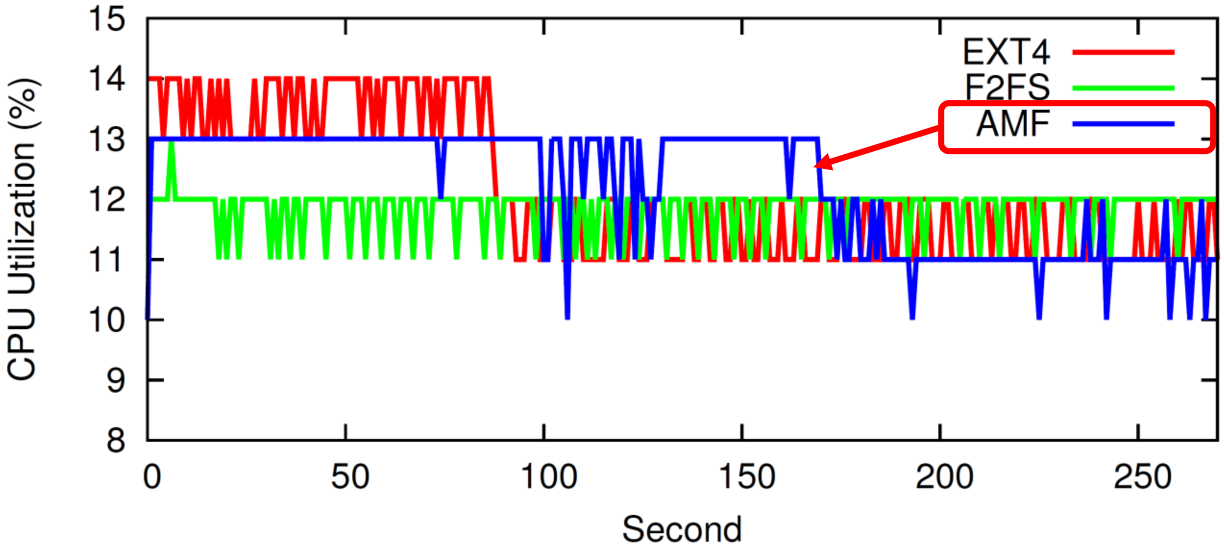
- AMF achieves 6% and 37% better lifetimes than EXT4 and F2FS, respectively, on average

Resource (DRAM & CPU)

- FTL mapping table size

SSD Capacity	Block-level FTL	Hybrid FTL	Page-level FTL	AMF
512 GB	4 MB	96 MB	512 MB	4 MB
1 TB	8 MB	186 MB	1 GB	8 MB

- Host CPU usage



Conclusion

- We proposed the Application-Managed Flash (AMF) architecture.
  - AMF was based on a new block I/O interface, called AMF IO, which exposed flash storage as append-only segments
  - Based on AMF IO, we implemented a new FTL scheme (AFTL) and a new file system (ALFS) in the Linux kernel and evaluated them using our in-house SSD prototype
  - Our results showed that DRAM in the flash controller was reduced by 128X and performance was improved by 80%

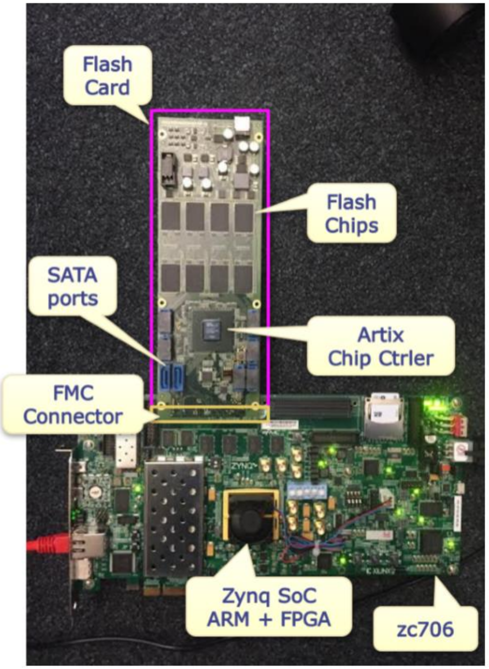
- Future Work
  - We are doing case studies with key-value stores, database systems, and storage virtualization platforms

Discussion

- Hardware Implementation of AFTL
- Smaller segment size
- Open-Channel SSDs vs AMF
- …

Hardware Implementation of AFTL

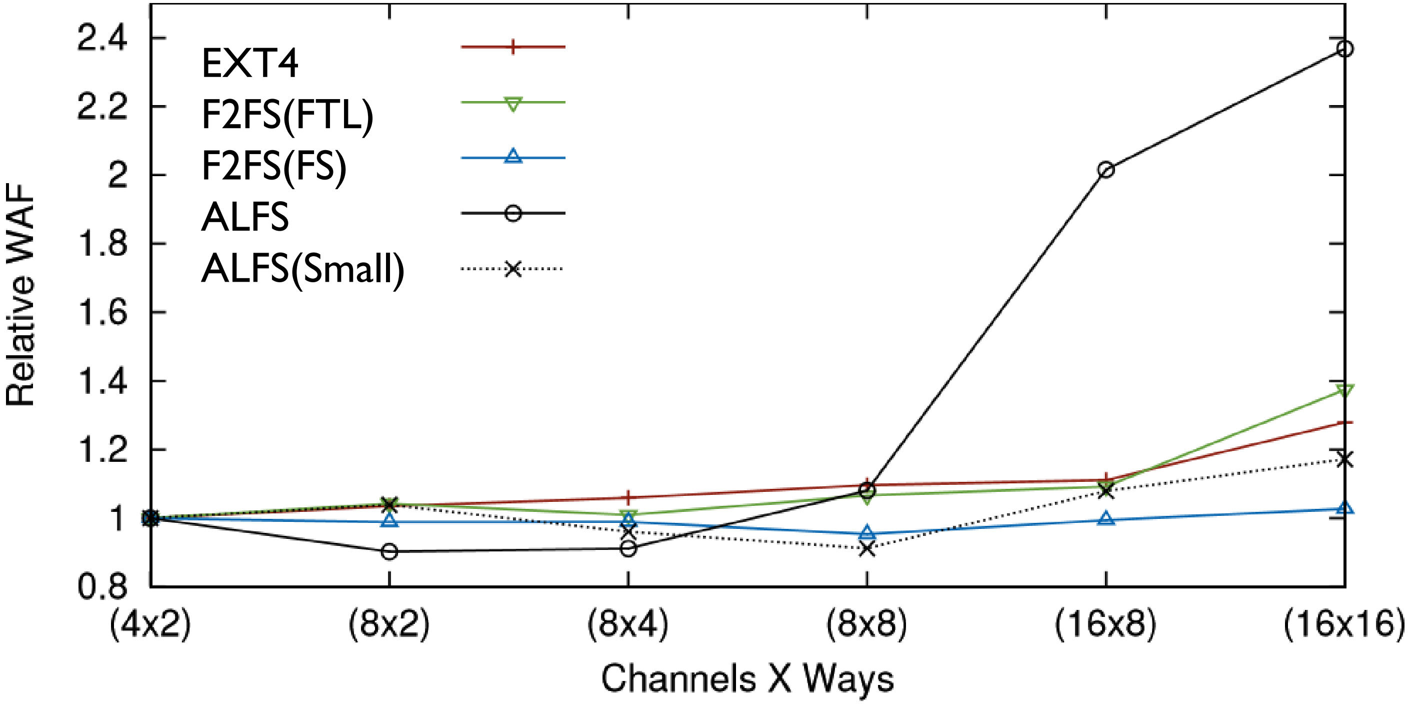
- Implement pure hardware-based FTL in FPGA that support the basic functions of AFTL
  - Expose block I/O interfaces to host
  - Segment-level remapping
  - Dynamic wear-leveling
  - Bad-block management
  - …



- It is still a proof-of-concept prototype
- But, it strongly shows that CPU-less and DRAM-less flash storage could be a promising design choice

Smaller Segments

- ALFS shows good performance with smaller segments
  - F2FS and ALFS(small) are with 2MB segments
  - The segment size of ALFS increase in proportional to channel and way #



Open-Channel SSDs vs AMF

- Two different approaches are based on similar ideas
- The main difference is *a level of abstraction*
  - AMF still maintains block I/O abstraction
  - AMF respects the unreliable NAND management by FTL
  - AMF allows SSD vendors to hide the details of their SSDs
  - AMF requires small modification on the host kernel side
  - AMF exhibits better data persistency and reliability
  - …

Source Code

- All of the software/hardware is being developed under the GPL license

- Please refer to our Git repositories
  - Hardware Platform: https://github.com/sangwoojun/bluedbm.git
  - FTL: https://github.com/chamdoo/bdbm_drv.git
  - File-System: https://bitbucket.org/chamdoo/risa-f2fs

Thank you!